

BUG-BYTE

SOFTWARE

ZXAS

MACHINE CODE ASSEMBLER

© COPYRIGHT 1981

This assembler uses 5K of RAM, and resides at the top of the memory, i.e. it uses locations 27648 to 32767. The RAMTOP system variable is automatically set to prevent loss of the assembler by overwriting with BASIC, or with the NEW command.

To use the assembler, simply LOAD it from cassette, using the filename ZXAS, then RUN the program. This transfers the assembler to the top of the memory. Lines 10, 20, and 30 can now be deleted, and the assembler is ready for use.

Standard Zilog mnemonics are understood by the assembler (with the exception of using full stops instead of commas, to aid keying in the mnemonics). The instructions are entered into the BASIC program inside REM statements, with more than one instruction per line being allowed if they are separated by semi colons. The entire assembly listings are enclosed in brackets, which are also in REM statements. Comments may be placed in the listings by typing *, followed by the comment.

As each instruction is processed by the assembler, there are various outputs to the television screen, a typical example being

100	408A	76	HALT
-----	------	----	------

This simply means that the instruction HALT occurs at line 100 of the listings, the machine code for HALT is 76 (hexadecimal), and that the instruction was assembled into location 408A (hexadecimal). This information is displayed, and when the screen is full you have to press a key for assembly to continue (pressing break at any time during assembly will terminate the program). Assembly will finish with an error message at the bottom of the screen; these having the following meanings;

- Error 0 - no mistakes
- Error 1 - no (found
- Error 2 - no) found
- Error 3 - illegal label
- Error 4 - illegal instruction
- Error 5 - number out of range
- Error 6 - relative jump out of range

To assemble the machine code, simply GO TO 9000, when you will be prompted for the starting location for the machine code (see later for typical values). Since not all of the instructions can be fully assembled immediately (e.g. jumps forward to a label), the assembler will process every instruction twice - a two pass assembly.

Numbers are assumed to be decimal, unless preceded by \$, when they are interpreted as hexadecimal. Thus the two instructions ADD A.34 and ADD A.\$22 are equivalent, with them both meaning add 34 (decimal) to the accumulator. Up to 256 labels are allowed, these being denoted by :L followed by the label number followed by the corresponding instruction. Thus the allowed labels range from :L0 to :L255. These labels are useful for jump and call instructions, and as temporary stores for data. This last use is illustrated by the following, which uses L30 as a "variable".

```
LD A.2
LD (L30).A
RET
:L30NOP
```

The label 30 has to be used to define the location initially, and must be filled with a dummy instruction. Instead of using labels with relative jumps, the displacement byte may be used, as long as it is preceded by + or - as appropriate. Thus the following two portions of code are equivalent:

i)	JR Z.+1	ii)	JR Z.L5
	INC HL		INC HL
	LD (HL).A		:L5LD (HL).A

As an example routine, type in the following, which will produce a delay of up to 25.6 seconds in steps of 0.1 seconds (the actual delay is determined by the value with which B is initialized with in line 20).

```
10 REM (
20 REM LD B.0;LD DE.$FFFF
30 REM :L1LD HL.14130;:L2ADD HL.DE
40 REM JR C.L2;DJNZ.L1;RET
50 REM )
```

At the end of the machine code, you should RETURN to BASIC. Pressing the break key will have no effect in a machine code routine; thus it is a good idea to test for this at various places. The code below will loop until the break key is pressed, when you will be returned to BASIC. Note that this uses a routine in Sinclair's ROM.

```
10 REM (
20 REM CALL$F43;JR C.-5;RET
30 REM )
```

Machine code programs often result in loss of the program, thus it is advisable to SAVE the program before the machine code is executed (this simply being done by a USR call to the beginning of the routine).

Memory is only SAVED as far as the memory location contained in the system variable E-LINE, thus unless the code is assembled into a part of the program, it will not be saved. The listings, however, are saved. If it is the machine code rather than the listings that you wish to save then the following procedure should be followed. Make the first line of the program a REM statement, and make the line long enough to accomodate all of the assembled code. Then assemble the code into location 16514 and onwards (location 16514 is the first character after REM), and delete all of the listings. The program may now be saved (the assembler is not saved by the SAVE command), with the machine code safely inside the program.

If there is no need to save the assembled code, then it may be assembled into the spare region of memory - STKEND points to this (see page 171 of the ZX81 manual). You should remember that if you come back to the listing program at a later date, the assembler should be in the ZX81, so follow this procedure. Load the assembler, run it, then type NEW, then load in the listings, when you are ready to reassemble the listings.

Unfortunately, space does not permit detailed comments on Z80 instructions, thus the user should consult books such as "The Z80 Instruction Handbook", or any text on Z80 programming. These sheets are only a description of how to use ZXAS, and are not a course in machine code programming.

THE MNEMONICS

Various abbreviations are used in the following list of instructions:

r is any single register, i.e. A, B, C, D, E, H or L
dis is a one byte displacement, in the range -128 to +127
data is immediate data, either one or two bytes as appropriate
mem is either (HL, (IX+dis) or (IY+dis). These latter two instructions take the value of designated index register, add on the displacement to obtain an address which is used in the instruction.

The first three letters of most instructions are important, so you must get these correct, especially the spaces.

ADC A.mem	ADC A.r	ADC A.data	ADC HL.BC
ADC HL.DE	ADC HL.HL	ADC HL.SP	ADD A.mem
ADD A.r	ADD A.data	ADD HL.BC	ADD HL.DE
ADD HL.HL	ADD HL.SP	ADD IX.BC	ADD IX.DE
ADD IX.IX	ADD IX.SP	ADD IY.BC	ADD IY.DE
ADD IY.IY	ADD IY.SP	AND mem	AND r
AND data	BIT 0.mem	BIT 0.r	BIT 1.mem
BIT 1.r	BIT 2.mem	BIT 2.r	BIT 3.mem
BIT 3.r	BIT 4.mem	BIT 4.r	BIT 5.mem
BIT 5.r	BIT 6.mem	BIT 6.r	BIT 7.mem
BIT 7.r	CALL address	CALLC.address	CALLM.address
CALLNC.address	CALLNZ.address	CALLP.address	CALLPE.address
CALLPO.address	CALLZ.address	CCF	CP mem
CP r	CP data	CPD	CPDR
CPI	CPIR	CPL	DAA
DEC mem	DEC r	DEC BC	DEC DE
DEC HL	DEC IX	DEC IY	DEC SP
DI	DJNZ.dis	EI	EX (SP).HL
EX (SP).IX	EX (SP).IY	EX AF.AF	EX DE.HL
EXX	HALT	IMO	IMI
IM2	IN r.(C)	IN A.port	INC mem
INC r	INC BC	INC DE	INC HL
INC IX	INC IY	INC SP	IND
INDR	INI	INIR	JP (HL)
JP (IX)	JP (IY)	JP address	JP C.address
JP M.address	JP NC.address	JP NZ.address	JP P.address
JP PE.address	JP PO.address	JP Z.address	JR dis
JR C.dis	JR NC.dis	JR NZ.dis	JR Z.dis
LD (address).A	LD (address).BC	LD (address).DE	LD (address).HL
LD (address).IX	LD (address).IY	LD (address).SP	LD (BC).A
LD (DE).A	LD mem.r	LD mem.data	LD A.(address)
LD A.(BC)	LD A.(DE)	LD r.r.	LD r.data
LD r.mem	LD A.I	LD A.R	LD BC.(address)
LD BC.data	LD DE.(address)	LD DE.data	LD HL.(address)
LD HL.data	LD I.A	LD IX.(address)	LD IX.data
LD IY.(address)	LD IY.data	LD R.A	LD SP.(address)
LD SP.data	LD SP.HL	LD SP.IX	LD SP.IY
LDD	LDDR	LDI	LDIR
NEG	NOP	OR mem	OR r
OR data	OTDR	OTIR	OUT (C).r
OUT port.A	OUTD	OUTI	POP AF
POP BC	POP DE	POP HL	POP IX
POP IY	PUSH AF	PUSH BC	PUSH DE
PUSH HL	PUSH IX	PUSH IY	RES 0.mem

RES 0.r	RES 1.mem	RES 1.r	RES 2.mem
RES 2.r	RES 3.mem	RES 3.r	RES 4.mem
RES 4.r	RES 5.mem	RES 5.r	RES 6.mem
RES 6.r	RES 7.mem	RES 7.r	RET
RET C	RET M	RET NC	RET NZ
RET P	RET PE	RET PO	RET Z
RETI	RETN	RL mem	RL r
RLA	RLC mem	RLC r	RLCA
RLD	RR mem	RR r	RRA
RRC mem	RRC r	RRCA	RRD
RST 00	RST 08	RST 10	RST 18
RST 20	RST 28	RST 30	RST 38
SBC A.mem	SBC A.r	SBC A.data	SBC HL.BC
SBC HL.DE	SBC HL.HL	SBC HL.SP	SCF
SET 0.mem	SET 0.r	SET 1.mem	SET 1.r
SET 2.mem	SET 2.r	SET 3.mem	SET 3.r
SET 4.mem	SET 4.r	SET 5.mem	SET 5.r
SET 6.mem	SET 6.r	SET 7.mem	SET 7.r
SLA mem	SLA r	SRA mem	SRA r
SRL mem	SRL r	SUB mem	SUB r
SUB data	XOR mem	XOR r	XOR data